**Лекция за 11.04.2012, часть 2**

***Синдром глупого окна (продолжение). Синдром быстрого старта. Алгоритм медленного старта. Устранение заторов. TCPV6. UDP.***

***Синдром глупого окна (продолжение)***

Здесь мы можем начать следующую передачу данных после прихода соответствующего ***ACK***-a. Пока все нормально: мы передали на 1001-й ***ACK*** порцию данных с 1801-ого по 2000-й октет, а дальше начинают приходить ***ACK*** -и на короткие сегменты. Сначала приходит ACK=1051, а потом- ***ACK*** =1201. ***ACK*** =1051 разрешает передать нам только 50 октет данных, потому что каждый раз до прихода ***ACK*** -a окно остается заполненным, поэтому мы вынуждены на ***ACK*** =1051 передать данные с 2001 по 2050. Следующим приходит ***ACK*** =1201, с него мы можем передать данные с 2051-ого октета по 2200-ый. Можно легко увидеть, что эта ситуация является *самовоспроизводящейся*. Теперь каждый раз с увеличением номеров октетов на тысячу будет воспроизводиться ситуация, в которой вы вынуждены передавать сначала всего 50 октет, а потом-150.

Это приводит к довольно существенному *уменьшению пропускной способности* *сети*. Более того, может не требоваться, чтобы кто-то досрочно передал короткий октет, вовсе не обязательно использовать флаг ***PSH*** для передачи сегмента в 50 октет. Если у вас размер окна 1050, а не 1000, то возникает ситуация, при которой вы вынуждены будете передать на некотором шаге данные с 1001-ого по 1050-ый октет, а далее будете передавать снова по 200 октет, с 1051-ого по 1250-ый и т. д. То есть ситуация воспроизведется на шаге 2051- 2100.

Оказывается, что дело даже не во флаге ***PSH***, беда может возникнуть из-за не очень удачного согласования длин окон и характерных размеров сегментов. Можно было бы сказать, что виноваты интерактивные протоколы, но если проблема возникает из-за неудачного выбора TSP-окна, это означает, что *проблема в логике самого* ***TSP*.**

С ***синдромом “глупого окна”*** столкнулись, когда появилась референсная реализация ***4BSD***. Пока дело происходило в пределах одного здания (комплекс Калифорнийского Университета, Санта-Барбара), где были быстрые каналы связи, малое количество узлов и, следовательно, небольшие времена задержки, эта ситуация не становилась заметной. Но когда подобные явления стали возникать на трансконтинентальных медленных линиях, скорость передачи данных по которым была 1мбит/с, ***синдром “глупого окна”*** стал портить жизнь очень многим

Был предложен достаточно простой ***алгоритм***, который удалось достаточно легко интегрировать в существующие TSP-стеки.

***Суть алгоритма заключается в следующем:***

1. *Принимающая сторона не должна анонсировать очень маленькие окна*. Говоря более конкретно, адресат не должен анонсировать размер окна, больший текущего, пока последний не может быть увеличен либо на размер максимального сегмента (***Maximum Segment Size, MSS***), либо на 1/2 размера буфера приема, в зависимости от того, какое значение окажется меньшим.

Это избавляет нас от *второй ситуации*, когда возникает небольшой кусок окна, в котором мы можем начать вести передачу.

Эта же ситуации могла возникнуть и *третьим способом*, если бы мы передавали данные с длиной окна 1000, передали 1000 октетов, а потом в первом же подтверждении удаленная сторона увеличила бы его до 1050.

Смысл выше написанного правила заключается в следующем: *вы не должны создавать ситуацию, при которой принимающая сторона провоцирует отправителя данных на отправку коротких сегментов.* Одна из таких провокационных ситуаций - это как раз увеличение размера окна на размер, который меньше максимального размера сегмента. На вторую часть про размер буфера приема пока можно не обращать внимания.

1. Отправитель должен воздержаться от передачи, пока он не сможет передать сегмент максимального размера или сегмент, размер которого больше половины максимального размера окна, который когда-либо анонсировался принимающей стороной.

*Как в этой ситуации действовать по первой предложенной схеме?*

Нам точно нужно передать 50 октетов - это сегмент меньший ***MSS***(здесь это 200), но мы понимаем, что когда мы их передадим, в следующий раз придется передать только 150 октетов. Но, так как нам все равно надо будет их передать, можно делать это не сразу, а дождаться момента, когда можно будет передать сразу 200 октетов, то есть дождаться следующего ***ACK***-а. Далее этот алгоритм будет сглаживать скачки от 200 до 50, чтобы как-то выровнять скорость передачи данных.

Вы можете сами провести некий *эксперимент*: запустить аську и попытаться отправлять по ней куски большого размера(***MSS*** в современном ***TCP***- 1496 октетов (размер поля данных ***Ethernet***)), а потом посмотреть, как это будет сказываться на пропускной способности линии. Для этого эксперимента надо хорошо подбирать канал связи, потому что на быстрых каналах связи все это сильно рассасывается. Подумайте об этом.

Протоколы, которые требуют коротких сегментов (это **Не понимаю слова!** 1.18.05 Messenger и удаленные ***Shell***-ы), страдают ***синдромом “глупого окна”*** на медленных линиях. Проблемы с ними разрешаются переходом на высокоскоростные линии, когда данные передаются практически в синхронном режиме, то есть, когда удается передавать и принимать подтверждения почти одновременно.

***Проблема быстрого старта***

Но это не единственная проблема ***TCP***. Следующая проблема, которая связана с управлением потоком в ***TCP***- это ***проблема быстрого старта***, которая стала актуальна почти сразу после появления стека ***4BSD***. Она заключается в следующем: если вы начинаете сразу передавать данные, пытаясь выбрать максимальный размер окна передачи, не дожидаясь подтверждения, то это, во-первых, приводит к ***взрывному росту трафика в сети***, а, во-вторых, провоцирует возникновение ***синдрома “глупого окна”***, что видно по предыдущей картинке. Мы начинаем передавать данные, и передаем, выбирая весь размер доступного окна передач, не дожидаясь подтверждения.

С ***взрывным ростом трафика*** все достаточно очевидно: если вы сразу начинаете передавать данные с большой скоростью, то у вас резко увеличивается объем передаваемых данных. Как говорилось в лекции по протоколам сетей глобальной связи, ***АТМ*** и *сети с коммутацией* обладают определенной инерционностью. Это значит, что если вам требуется дополнительная полоса пропускания, то она не может быть сразу выделена. То есть должно пройти некоторое время, пока аппаратура глобальной сети адаптируется к увеличению трафика.

В качестве *эксперимента*, возьмите ***Windows 95***, поставьте на нее современный ***Mozilla Firefox*** и откройте одновременно 5-6 вкладок. Или просто запустите пять экземпляров ***Internet Explorer*** на канале в 10 Мбит, который висит на ***ADSL***, или, желательно, на ***ADSL*** + ***ATM***.

/\*Кстати, когда вам говорят, что ваш тариф ADSL 10мбгит, это совсем не значит, что вам все время держат полосу пропускания 10 Мбит. Это означает, что скорость выше 10 Мбит вы не получите, а полоса делится более-менее равномерно между всеми абонентами. И пока у вас средний трафик идет 2 мегабита в секунду, вам и будут выделяться эти два мегабита. А чтобы перейти с двух мегабит на 10, потребуется определенное время релаксации, небольшое, порядка нескольких десятков секунд, но оно есть, и именно в процессе релаксации возникают заторы. \*/

С ***проблемой быстрого старта*** столкнулись при начале взрывного роста интернета и резкого увеличения числа пользователей. Стало ясно, что с ней надо как-то бороться.

*Самый простой вариант*, который был реализован почти немедленно, заключался в следующем: при начале передачи данных, сразу после установления соединения, разрешалось передавать данные не до *полной выборки* анонсированного окна передачи, а до *выборки размера его половины*. Если размер анонсированного окна передачи 1000 октетов, то можно было свободно передавать данные до выборки 500 октетов. После этого обязательно надо было дождаться подтверждения и дальше передавать данные.

***Логика такова***: есть данные по 100 октетов, вы передали 500 октетов, после этого получили подтверждение на первую сотню. Допустимое окно передачи- 1000 октетов, 500 октетов передали, окно сместилось направо на 100 октетов, осталось 600, из которых можно передать 300. Передали 300, дождались следующего подтверждения, которое пришло на вторую сотню октетов, окно сдвинулось направо еще на одну позицию. Доступный размер для передачи данных стал 400 октетов. Из них передаете 200 и ждете подтверждения на третью сотню, смещаетесь вправо еще на одну позицию, получается 300 октетов. При дальнейших не целых значениях применяются более тонкие стороны алгоритма. ***То есть идея заключается в следующем****:* *нельзя увеличивать скорость передачи данных, до того как более-менее не произойдет релаксация соединения.*

Современные реализации алгоритма быстрого старта работают с величиной под названием ***окно переполнения***.

***Некоторое отвлечение****:* вернемся к обсуждению опций ***IP*** (поле опций в заголовке ***IP-датаграмм***). Одна из опций ***IP***- это *возможность задания размера* ***MSS***.

Если сторона либо устанавливающая соединения, либо отвечающая на установление соединения абсолютно достоверно знает, что при приеме данных, она не может работать со стандартным ***MSS***, который составляет1496,или 1492, или 1500 октетов в зависимости от используемого сегмента ***Ethernet***, она может установить опцию ***MSS*** и передать в нем желаемый размер сегмента. То есть если этот ***MSS*** выставлен в опциях, то удаленной стороне рекомендуется придерживаться его же. Это впервые появилось при *борьбе с фрагментированием*.

Вторая причина, почему надо указывать ***MSS***,- это тот случай, при котором вы используете какой-нибудь хитрый протокол типа ***ASDM*** для подключения к **чему?? 1.29.15,** в котором ***MSS*** может быть коротким. Также это может вызываться чем-то, влияющим на фрагментацию ***IP***, или проблемами, которые могут быть связаны с flow control. Малый ***MSS*** также может быть, если вы используете модем на достаточно зашумленных линиях и не можете применять высокоскоростные протоколы. *Таким образом, значение* ***MSS*** *можно задать.*

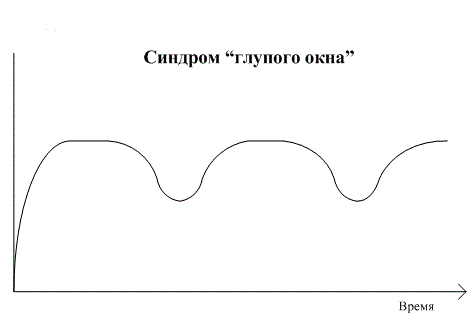
Вернемся теперь к ***окну переполнения***. ***Окно переполнения*** при установлении соединения выставляется в значении ***MSS*** (либо в стандартное, либо в установленное адресатом при установлении соединения). Если адресат не устанавливает специального значения ***MSS***, но вы по каким-то априорным соображениям знаете, что либо линия, по которой вы с ним соединяетесь, имеет *большие задержки*, либо низкую *пропускную способность*, то ***окно переполнения*** выставляется в какое-либо *маленькое число*. В зависимости от реализации стека это 512 или 536 октетов. При вычислении ***доступного окна*** отправитель использует меньшее из *предлагаемого окна* и *окна переполнения*. Каждый раз, когда отправитель получает подтверждение полученного сегмента, его ***окно переполнения*** увеличивается на величину этого сегмента.

Этот алгоритм очень близок к тому, который был описан сначала, но является немного более гибким. Он лучше справляется с проблемами, когда размер максимального допустимого окна передачи сокращается до размера где-то 100 октетов.

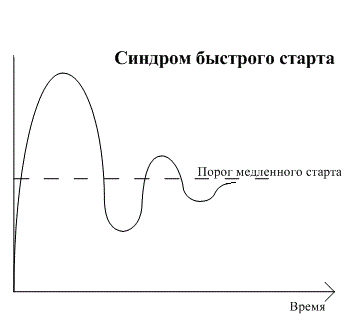
***Пропускная способность линии***

Теперь посмотрим, как меняется *реальная пропускная способность линии* при изученных нами проблемах.

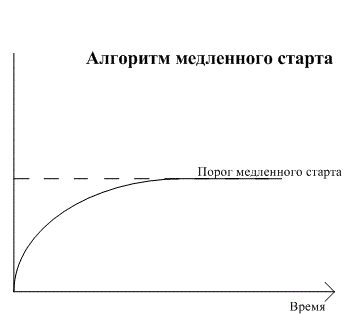
Когда мы встречаемся с ***синдромом “глупого окна”***, *пропускная способность линии* имеет такую форму:



В случае ***синдрома быстрого старта*** пропускная способность меняется так:



При применении ***алгоритма медленного старта*** получается следующее:



Асимптота та же самая, нет выстрелов вверх.

***Устранение заторов:***

***Переполнение*** или ***затор***, может возникнуть в сети по многим причинам. Самая простая - это *падение пропускной способности линии* *в короткие моменты времени*, происходящее либо в результате каких-либо эксцессов на канале связи, либо в результате алгоритма работы протокола глобальной сети, либо *если вы плюнули в линию данных больше, чем она сможет переварить*. ***Синдром быстрого старта*** является частным случаем ***проблемы заторов***.

***Алгоритм***, который помогает избежать *заторов,* достаточно сложен. Он появился во второй половине 90-ых годов, когда качество линий и скорость, были довольно большими. ***Алгоритм*** основывается на предположении, что *потеря данных*, вызванная ошибками передачи по физической среде, *пренебрежимо мала* (гораздо меньше 1%). На самом деле, если у вас линия такова, что на ней теряется примерно 10% ***TSP***-сегментов по разным причинам, например, они доходят с кривой контрольной суммой или просто не доходят, то на такой линии работать, как правило, невозможно. Поэтому, предположение о грани в 1% довольно разумно. *Если линия работает, то уровень потерь на ней не очень высок*. Иначе, потеря данных свидетельствует не о каких-то проблемах линии, а о ***возникновении затора***.

Отправитель данных может судить об этом либо при *существенных паузах в подтверждении получения*, либо при получении *дубликатов подтверждения*. Если время прохождения сегмента не превышает двух ***MSL***, то вы успеваете получить подтверждение. Если время резко вырастает, то это - признак ***затора***.

*Дубликаты подтверждений* означают, что подтверждение первой попытки доставить данный ***TSP***-сегмент пришло уже после срабатывания *ретрансмит - таймера*. Данные были доставлены, но время доставки данных резко превысило значение два ***MSL***. Стоит напомнить, что время ***MSL*** меняется: если медленно падают параметры линии, время ***MSL*** увеличивается.

***Алгоритм устранения заторов***

Для работы ***алгоритма*** требуется два параметра: ***окно переполнения*** ***CWND*** и ***порог медленного старта*** ***SSThresh***. *Этот алгоритм перекрывает алгоритм медленного старта*.

1. Начальное значение ***CWND*** задается равным размеру одного сегмента

* ***CWND=MSS***

Начальное значение ***SSThresh*** задается равным ***максимально возможной длине***

***сегмента*** (***MaxInt*** минус длина заголовка)

* ***SSThresh=MaxInt***

1. ***Максимальное количество данных***, которое мы можем одновременно передать отправителю без получения подтверждения (***MaxDataSize***), равняется минимуму из ***CWND*** и ***SSThresh***

* ***MaxDataSize=min(CWND, SSThresh)***

В начале, пока ***SSThresh*** инициализирован большим числом, ***MaxDataSize*** равен ***CWND***, что является показателем того, что будет работать *алгоритм медленного старта*

1. При возникновении ***затора*** параметр ***SSThresh*** устанавливается равным максимуму из ***размера текущего окна передачи*** ***(WNDSz***) и размера двух сегментов.

* ***SSThresh=max(WNDSz,2MSS)***

То есть размер ***SSThresh*** может резко упасть от значения ***MaxInt*** до значения максимума.

Если затор детектируется по тайм-ауту, то дополнительно размер ***CWND*** устанавливается равным половине ***разрешенного окна передач*** (***AcWNDSz***)

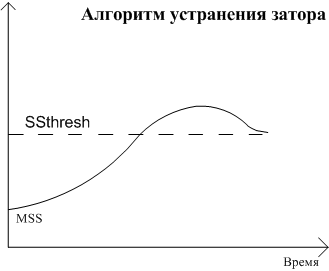
* ***CWND= AcWNDSz***

Именно этот случай работает при *медленном старте*

1. Когда отправитель получает подтверждение, он увеличивает размер ***cwnd*** на размер подтвержденных данных в полном соответствии с *алгоритмом медленного старта*. В случае если не повторяются задержки с тайм-аутами, размер ***SSThresh*** устанавливается на ***MaxInt.***

*Этот алгоритм позволяет довольно быстро, быстрее, чем в алгоритме медленного старта, наращивать скорость передачи данных в начале.*

*Пропускная способность* канала выглядит так:



На рисунке видно, что *порог медленного старта* сильно сглажен, его максимальная высота- ***MSS***, что дает практически *оптимальную пропускную способность* при установлении соединения. Это позволяет достаточно быстро нарастить скорость передачи данных и не приводит к возникновению заторов.

*Оптимальность алгоритма* заключается в том, что за достаточно короткое время вы удается получаете возможность выйти на *предельный уровень пропускной способности*. Минимальное время в данном случае- время оправки нескольких десятков тсп-сегментов. ***Алгоритм медленного старта*** выходит на пропускную способность, отличающуюся на 1% от теоретически возможной, на интервале в 50-100 сегментов. ***Алгоритм устранения заторов***- примерно на 20-30.

***Разница между алгоритмом медленного старта и алгоритмом устранения затора заключается в следующем:*** в *алгоритме медленного старта* мы быстро начинаем, но при этом все время себя удерживаем, пытаемся подобраться к *асимптоте* снизу и нащупать ее. В ***алгоритме устранения затора***, мы намеренно выскакиваем выше, понимаем, где она расположена и стремимся быстро вернуться к ней сверху. ***Алгоритм устранения затора*** - это в чистом виде *реализация алгоритма наискорейшего спуска* (в самом простом случае - алгоритм численного нахождения экстремума).

Все эти ухищрения не дают стопроцентной гарантии устойчивости ***TSP***- соединения. Причина очень простая: когда вы используете управление потоком с ***TSP***, в некоторых ситуациях вы вынуждены будете передавать данные, не смотря на то, что знаете, что это приведет к возникновению ***затора***. Существуют тайм-ауты закрытия соединения по неполучению данных. Таймер ретрансмитта, как правило, ***4MSL***,значит счетчик ретрансмиттинга- 5-7 попыток ретрансмита. Соответственно, так как 5\*4=20, 7\*4=28, в этом промежутке лежит *тайм-аут закрытия соединения*. Если за это время не пришли никакие данные, то сторона не старается дальше держать ***TSP***-соединение открытым. Для того чтобы избежать разрыва соединения по тайм- ауту, вы вынуждены передавать данные.

***IPV6***

***Алгоритм устранения затора*** может решить 90% проблем с управление потоком. Оставшиеся 10% приходятся на резкую *деградацию соединения*, либо по физическим причинам (например, параллельно было два оптических кабеля, из которых один вышел из строя, и скорость упала в два раза), либо из-за неудачного переключения маршрутизации, либо из-за выхода из строя какого-нибудь магистрального оборудования. То есть это резкое, но не кратковременное падение пропускной способности.

Чтобы бороться с такими ситуациями, нужны *более сложные механизмы управления потоком*, которые допускают временное приостановление передачи данных без разрыва соединения. Строго говоря, в классическом ***TSP*** такого механизма нет. Сначала он появился в ***TCPV6***- это ***TSP*** с надстройками на ***IPV6***, потом в ***IPV4***, и сейчас реализован во многих TSP-стеках (в Linux, FreeBSD, Windows7,Server2008). Этот механизм вводит два дополнительных флага, которые добавляются перед остальным полем флагов***: флаг*** ***CWR (Congestion Window Reduce)*** и ***ECE (ECN Echo)***. ***CWR*** – это описанное выше *окно переполнения*.

**Заголовок** ***TCPV6***

|  |
| --- |
| CWR |
| ECE |
| URG |
| ACK |
| PSH |
| RST |
| SYN |
| FIN |

Флаг ***ECE*** показывает, что данный узел поддерживает явное *уведомление о перегрузке.* Собственно, ***ECN*** и есть уведомление о перегрузке. Этот флаг используется *двояко*. Во-первых, он должен быть установлен стороной, которая поддерживает это уведомление о перегрузке, при установлении соединения. То есть, если с флагом ***SYN*** посылается флаг ***ECE***, это означает, что данная сторона поддерживает *расширенное управление потоком*. Соответственно, в дальнейшем, если эта сторона выставит в одном из своих ***TSP***-сегментов флаг ***ECE***, то это означает, что от принимающей стороны требуется приостановить передачу данных без разрыва соединения до тех пор, пока от стороны, выставившей флаг, не придет ***TSP*** -сегмент со снятым флагом ***ECE***. *Использование флага* ***ECE*** *позволяет приостановить передачу данных без разрыва соединения.* Это не единственное использование этого флага, в ***TCPV6*** он используется очень активно и для других целей.

Флаг ***CWR*** информируется точно так же: он должен быть установлен при установлении соединения, для того чтобы дать удаленной стороне понять, что исходная сторона этот флаг поддерживает. ***Смысл этого флага таков****:* если мы посылаем на удаленную сторону этот флаг, то мы просим удаленную сторону *уменьшить анонсированное окно передачи*. Это означает, что мы не может принимать данные со скоростью, с которой удаленная сторона их передает.

***Отличия*** в использовании флага ***CWR*** от работы с ***анонсированным окном передачи*** следующие: мы уменьшаем ***анонсированное окно*** для тех данных, которые мы ожидаем с удаленной стороны, при установке же ***CWR*** мы предлагаем удаленной стороне самой уменьшить окно передачи. Оно уменьшается в соответствии с *алгоритмом устранения заторов*. *Использование этих двух флагов реально улучшает ситуации, связанные с серьезной деградацией каналов.*

***Протокол UDP(User Datagram Protocol).***

В отличие от ***TSP***, ***UDP*** не поддерживает целостности данных и не устанавливает виртуальные соединения. Его функциональность ниже, чем у ***TSP***, но в результате этого он имеет *крайне низкие задержки при передаче*. При использовании ***UDP*** можно добиться скорости близкой к теоретической физической пропускной способности канала. Поэтому протокол ***UDP*** используется там, где требуется передача данных в реальном времени, и можно пожертвовать их целостностью. В первую очередь, это передача потокового видео и аудио, IP-телефония и т. п.

***Заголовок UDP***

0 15 31

|  |  |
| --- | --- |
| Source Port | Destination Port |
| Length | CRCR **тут надо еще слово, но у меня непонятно написано(** |
| Data | |

***Заголовок UDP-датаграммы***состоит из четырех полей и имеет фиксированную длину. Сначала идут поля ***Source Port*** и ***Destination Port***. Пространство портов в ***UDP*** точно такое же, как и в ***TSP***, то есть от нуля до ***MaxInt***, но порты ***TSP*** и ***UDP*** не пересекаются. Таким образом, можно иметь службы, настроенные на ***TSP***, ***UDP*** порты с одним и тем же номером. Следующее поле заголовка - это ***Length***, длина датаграммы. Последнее поле - это 16-битная контрольная сумма псевдозаголовка ***UDP***.

***Псевдозвголовок UDP***

0 31

|  |  |  |
| --- | --- | --- |
| Source IP | | |
| Destination IP | | |
| 0 | Data  17 | Length |

***Псевдозаголовок*** строится точно так же как и ***псевдозаголовок TCP***. Он состоит из трех полей, которые берутся из заголовка ***IP***-датаграммы (***Source IP, Destination IP*** и номер протокола, который для ***UDP*** равен 17) и одного поля, который берется из заголовка ***UDP***-датаграммы (***Length***). Для выравнивания в начало третьего слоя ***псевдозаголовка*** добавляется выравнивающее поле. *Таким образом, получается псевдозаголовок, который вычисляет контрольную сумму*.